FT EIHub

INTEGRATION DATA MAPPING (DCS for FTPS)

RELEASE 5.03.00

TECHNICAL GUIDE

PUBLICATION FTEIDC-RM005C-EN-E-JULY-2024
10007935820/PUB
Supersedes publication FTEIDC-RM005B-EN-E

Rockwell Automation

Contact Rockwell See contact information provided in your maintenance contract.

Copyright Notice

Trademark Notices

Other Trademarks

Warranty This product is warranted in accordance with the product license. The product's performance may be affected by system configuration, the application being performed, operator control, maintenance, and other related factors. Rockwell Automation is not responsible for these intervening factors. The instructions in this document do not cover all the details or variations in the equipment, procedure, or process described, nor do they provide directions for meeting every possible contingency during installation, operation, or maintenance. This product's implementation may vary among users.

This document is current as of the time of release of the product; however, the accompanying software may have changed since the release. Rockwell Automation, Inc. reserves the right to change any information contained in this document or the software at any time without prior notice. It is your responsibility to obtain the most current information available from Rockwell when installing or using this product.

Industry Terminology Rockwell Automation recognizes that some of the terms that are currently used in our industry and our publications are not in alignment with the movement toward inclusive language in technology. We are proactively collaborating with industry peers to find alternatives to such terms and making changes to our products and content. Please excuse the use of such terms in our content while we implement these changes.

Contents

Figures

Chapter 1

Read Me First

Audience and Expectations

This guide is intended for experienced professionals who understand their company's business needs as well as the technical terms and software dependencies described in this guide.

This guide provides an overview on data transformations used in the Enterprise Integration Hub (EIHub). The audience is assumed to be experienced users of Eclipse and Camel. As these are third-party products, this guide does not include detailed information about the operation and configuration of these products except information that is sufficient for these products to work within the context of EIHub.

For details, see the Apache Camel User Manual [C1] (page 3).

This guide also assumes that the supporting network equipment and software, including FactoryTalk EIHub, JMS provider, FactoryTalk® ProductionCentre® (FTPC), FactoryTalk® PharmaSuite® and related databases, have been installed.

Typographical Conventions

This documentation uses typographical conventions to enhance the readability of the information it presents. The following kinds of formatting indicate specific information:

Bold typeface	Designates user interface texts, such as ■ window and dialog titles ■ workflow and widget names ■ menu functions ■ panel and tab titles ■ box labels ■ object properties and their values (e.g. status).
[Text in square brackets]	Designates button names.

Italic typeface	Designates technical background information, such as ■ path, folder, and file names ■ methods ■ classes.
CAPITALS	Designate keyboard-related information, such as ■ key names ■ keyboard shortcuts.
`Monospaced typeface`	Designates code examples.

Organization

This book contains the following chapters:

- Overview (page 5)
  Provides an overview of EIHub functionality to create routes and perform custom data mapping.
- Using Eclipse IDE (page 11)
  Describes details regarding using Eclipse IDE.
- Implementing DCS Adapter Integration Touchpoints (page 17)
  Describes how data mapping is performed using the DCS integration touchpoints.
- Externalization of Configuration (page 47)
  Describes the details regarding the externalization of configuration.

Reference Documents

EIHub Documentation

The following documents are distributed with the EIHub installation or available on the Rockwell Automation Download Site.

No.	Publication	Part Number
A1	FT Enterprise Integration Hub Technical Guide Installation and Configuration	FTEI-IN005D-EN-E
A2	FT Enterprise Integration Hub Release Notes	FTEI-RN005D-EN-E

FTPC/FTPS Documentation

The following documents are distributed with the FTPC and FTPS installation or available on the Rockwell Automation Download Site.

No.	Publication	Part Number
B1	FT PharmaSuite Supported Platforms Guide	PSPG-RM011C-EN-E
B2	FactoryTalk ProductionCentre Technical Guide FTPC Installation	PCJB-IN011D-EN-E
B3	FT PharmaSuite Technical Guide DCS Adapter	DCTMAD-GR004A-EN-E

Third-party Documentation

The following third-party documentation is available online as reference.

No.	Document Title / Web Site
C1	Apache Camel User Manual (https://camel.apache.org/manual)
C2	Apache ActiveMQ Getting Started Document (https://activemq.apache.org/getting-started.html)
C3	Hawtio Documentation (https://hawt.io/docs/)

Chapter 2

Overview

> **TIP**
>
> This guide assumes that you are familiar with the Apache Camel software. If you are new to this product, before continuing with this guide, see the Apache Camel User Manual [C1] (page 3).

EIHub Functional Architecture

EIHub is used to integrate data between FTPC and an external system (for example, an ERP system such as a file system, a Distributed Control System (DCS), a LIMS system, etc.). The out-of-box installation of EIHub for PharmaSuite DCS integration provides MES to DCS integration touchpoints implemented using the DCS Adapter interface component.

Some integration tools provided by DCS, and third-party vendors use RESTful services, Web Services, COM-based services, and/or database access. The DCS Adapter component provides the MES solution with a DCS vendor-neutral interface based on XML payloads and Java Message Service (JMS) message queues. For details, see the "FT PharmaSuite Technical Guide DCS Adapter" [B3] (page 3). This allows sending standardized requests between the MES and any DCS as well as receiving standardized replies based on XML payloads. The corresponding XML schema is compliant with the S88/S95 standards and based on the B2MML standard schema.

PharmaSuite DCS Integration Scenarios Using the DCS Adapter

The following figure depicts the overall functional architecture of EIHub being used with DCS Adapter XML payloads as applied to the out-of-box PharmaSuite DCS integration touchpoints. For these integration touchpoints, EIHub processes DCS Adapter XML documents generated from the MES and posted on well-defined JMS message queues. The XML documents are transformed into the DCS-relevant message formats and uses DCS-specific interfacing technologies.

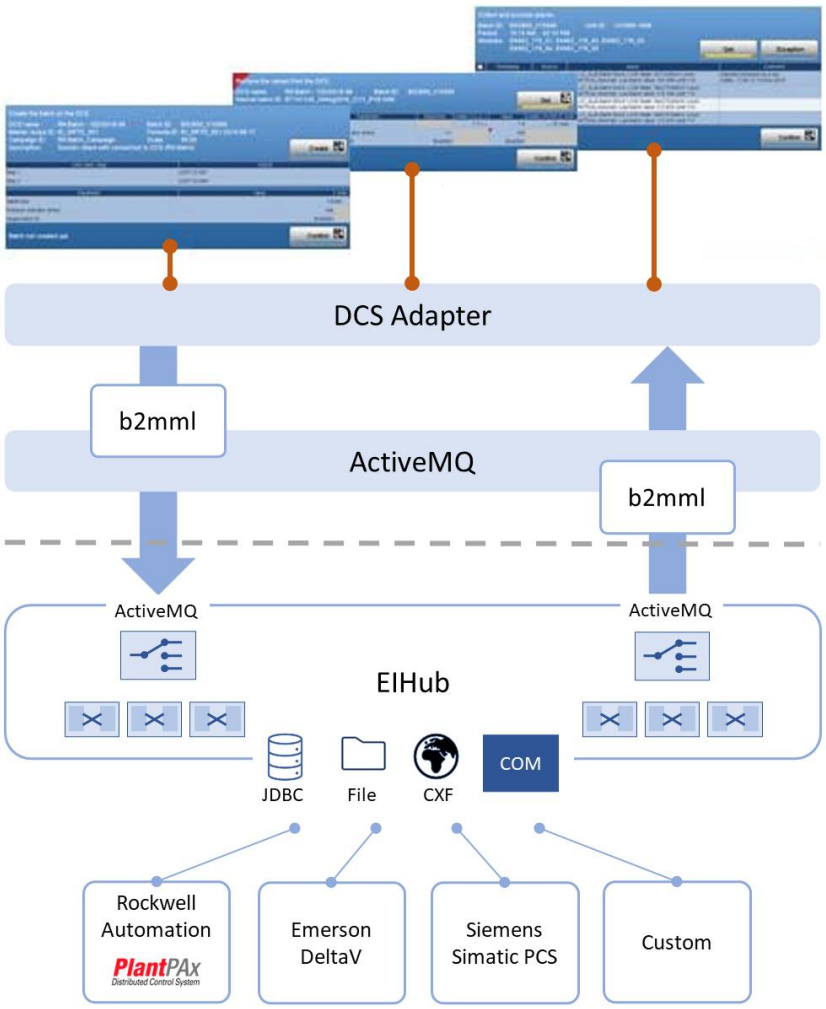

Figure 1: EIHub functional architecture - DCS adapter XML documents

The DCS Adapter typically runs within the MES layer. The MES client calls the DCS Adapter to communicate with the DCS system. The DCS Adapter creates and sends a request via the JMS communication layer.

EIHub communicates with the MES via JMS and with a DCS via the DCS-specific technology. EIHub adopts, translates, and routes the defined MES requests to the specific DCS according to their interface specification. For the communication, a DCS can use different technologies.

EIHub performs the following tasks:

- Listens to the requests of the DCS Adapter.
- Translates the received requests into the language of a specific DCS and communicates with the specific DCS according to the corresponding interface.
- Translates the replies of the DCS into the defined reply message for the DCS Adapter and sends the replies.

> **TIP**
> The DCS-specific processing part in EIHub will always require integration work.

Understanding the DCS Integration Touchpoints

This section is divided into the following sections:

- Pharma DCS Integration Touchpoints (page 7)
- PlantPAx Integration Touchpoints Using the DCS Adapter (page 8)

Pharma DCS Integration Touchpoints

To provide out-of-box functionality for the PharmaSuite DCS integration, Rockwell Automation has developed and packaged the following three DCS Adapter integration touchpoints which are operational using PlantPAx FactoryTalk (FT) Batch with a well-defined configuration setup:

- **Create DCS Batch (PlantPAx operational)**
  Creates a new batch in the DCS for a particular DCS master recipe and a set of parameters. This is typically triggered from the MES when a specific part of the MES recipe is to be executed automatically by a DCS.
- **Get DCS Alarms (PlantPAx operational)**
  Receives GMP-relevant alarm events from the DCS. This is typically done to document the alarms as MES exceptions. Therefore, it uses the review-by-exception features of the MES.
- **Get DCS Batch Values (PlantPAx operational)**
  Reads the batch report values from the DCS. This is typically done to re-use the values in MES processing/calculations or to add them to the MES batch report.

Groovy scripts are generally used during data transformation to process the incoming B2MML message and generate the B2MML response message.

PlantPAx Integration Touchpoints Using the DCS Adapter

The integration touchpoints Create DCS Batch, Get DCS Alarms, and Get DCS Batch Values are available for the PlantPAx DCS integration with PharmaSuite. For these integration scenarios, EIHub processes the incoming B2MML documents generated from PharmaSuite and posts the documents on well-defined ActiveMQ queues. The queue names follow a common naming scheme:

`DCSRequest_`<*LogicalDCSName*>`_`<*RequestType*>

For the PlantPAx integration touchpoints, the following three queues are defined:

PlantPAx Integration Touchpoint Queues

Integration Touchpoint	Queue Name
Create DCS Batch	DCSRequest_<DCSTargetSystem>_CreateDCSBatch
Get DCS Alarms	DCSRequest_<DCSTargetSystem>_GetDCSAlarmEvents
Get DCS Batch Values	DCSRequest_<DCSTargetSystem>_GetDCSBatchValues

DCSTargetSystem is equal to PlantPAx. This property is configurable in the *EIHub-app-dcs-<FTEI version>.0xx\config \application-dcs.properties* file.

The Camel Context file named *pharmasuite_plantpax_msb.xml* contains three routes that implement the integration for the three specific PlantPAx DCS integration touchpoints. The routes define all the processing and transformation that take place within EIHub (Camel) from the time a B2MML message is received on the message queue to when a B2MML response message is posted on the response queue.

Message transformation is implemented using dedicated Groovy scripts that consume the B2MML payload and generate the required data to exchange information between EIHub and the PlantPAx DCS.

When creating a DCS batch, the incoming B2MML message is transformed into an HTTP request, and this request is sent out to a RESTful FT Batch server implementation for PlantPAx Batch. The response from the PlantPAx RESTful FT Batch server is a JSON file, which is processed by a marshaller and a Groovy script into a valid B2MML response message.

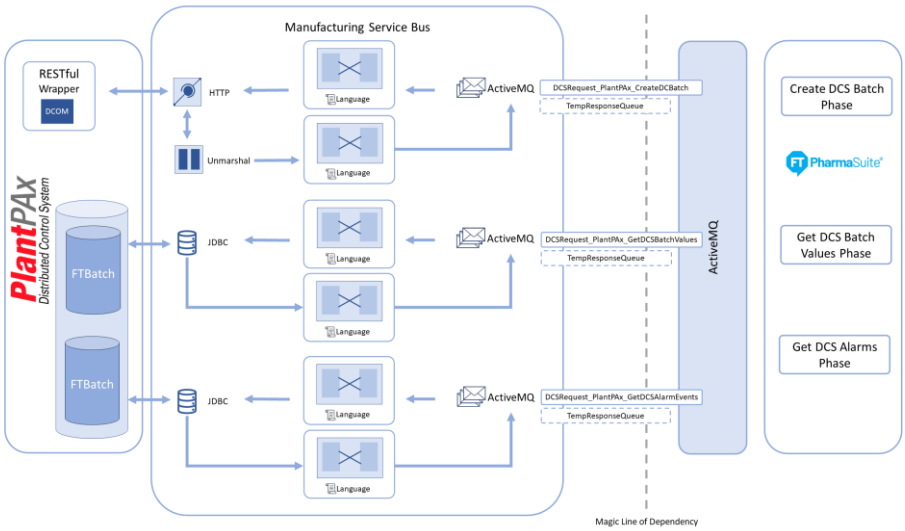

Figure 2: EIHub functional architecture - DCS adapter XML documents (PlantPAx)

When retrieving the Batch Alarms or Batch Report parameters (Get DCS Batch Values), dedicated routes transform the incoming B2MML message that was read on the appropriate JMS queue into a valid SQL request. The returned result set is transformed into a B2MML message and posted on the response queue.

For the Alarm values, the implementation requires a 1:1 view of the FT Alarms and Events table ConditionEvent in the BatchHistory Database.

When an exception occurs during the execution of the route and within the scope of the route exchange, the exception is handled, and a B2MML reject message is generated as a response.

Transformation and Data Mapping

MES integration between a DCS or a dedicated machine (e.g., Automated Weighing System) and PharmaSuite using the DCS Adapter interface involves transforming B2MML DCS Adapter request messages into DCS-specific requests and vice versa. It also means using the appropriate message queue names based on the DCS Adapter naming scheme for which a common pattern exists:

`DCSRequest_`<*LogicalDCS/MESName*>`_`<*RequestType*>

For details, see the "FT PharmaSuite Technical Guide DCS Adapter" [B3] (page 3).

Using Groovy Data Transformation and Mapping

Within EIHub, data transformations can be implemented by Camel based Java processors, pure Java Beans independent of the Camel APIs, and Groovy scripts. For the B2MML messages of the DCS Adapter interface, Groovy scripts are used. The XMLSlurper class is used to parse the XML payloads and the MarkupBuilder class is used to generate XML payloads.

The DCS Adapter XML request document transformation and mapping is dependent on the RequestType and the DCS being integrated with. In some cases, the XML document is transformed into a SQL query and database access is used to interface with the DCS. In other cases, the XML document is transformed into a JSON structure and a RESTFul service call is used to interface with the DCS.

Integration Tasks

Each MES DCS integration project using the DCS Adapter is unique and may require different approaches. However, the general approach used can be broken into the steps listed in this section. Note that these may or may not apply to your integration project.

The following steps offer a general guideline to implementing your integration project for the MES DCS integration touchpoints.

1. Understand the XSD schema of the DCS Adapter for the integration touchpoints (RequestType) you need to implement. For details, see the "FT PharmaSuite Technical Guide DCS Adapter" [B3] (page 3).
2. Determine the interface protocol for your specific integration touchpoint that is specific to the used DCS and determine which Camel component can be used. Remember that you can always develop your own Camel component or use Java Beans.
3. Understand the routes based on the DCS Adapter component [i.e., the MES always uses a JMS-based (ActiveMQ) interface]. This means that there are routes that read from JMS message queues (MES initiates the request) and routes that post messages on message queues (DCS initiates the request).
4. Use or extend the out-of-box PharmaSuite PlantPAx integration touchpoint mappings. See Implementing DCS Adapter Integration Touchpoints (page 17). Design your data mappings using Camel ID.

   Create the Java or Groovy code that implements the mapping of each integration touchpoint using Eclipse IDE. In some cases, it may make sense to use libraries like JAXB and Dozer to implement data transformations. If pre-processing transformation is required, develop the transformation, and include it in the Camel XML configuration files.
5. Deploy the mappings to your EIHub production environment.

Chapter 3

Using Eclipse IDE

> **TIP**
>
> Please note that EIHub package does not include Eclipse IDE. Any available Eclipse version that supports Java 17, such as 2023-12 (4.30.0), is recommended.

Creating Workspace Using Distributed Sources

The distributed source codes are in the *<EIHub_Install>\EIHub\installers\EIHub\Source-Code* directory. To setup the project workspace with the distributed source codes, proceed as follows:

1. Create a directory to place a source code.

   > **TIP**
   >
   > For details on setting up the project structure, see instructions included in the readme file: *<EIHub_Install>\installers\EIHub\Source-Code* directory.
   > where *<EIHub_Install>* is the directory where the artifact archive is extracted, e.g., *C:\Rockwell_install*.

2. Open Eclipse workspace, import the extracted sources as a Maven project.
   The project structure should look similar as the following:

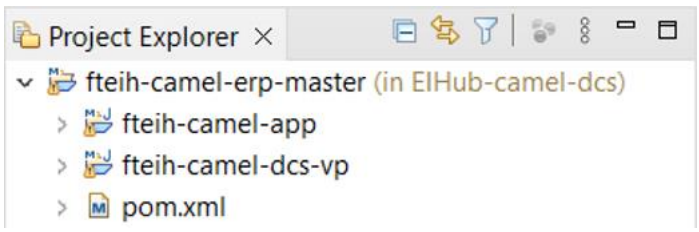

Figure 3: Eclipse project structure

> **TIP**
>
> Custom Java files can be added to the fteih-camel-dcs-vp project.

3. Update the following configuration files in the **fteih-camel-app** project:

   - The application *.property* files:
     - *application.properties*
     - *application-<valuepack>.properties*
     - *custom-application-<valuepack>.properties*

   where <valuepack> can be *discrete*, *process*, *dcs*, or *erp*.

   Please refer to the out of the box application property files to add all necessary configuration properties in the *custom-application-<valuepack>.properties* file.

TIP
Do not edit property files related to the other value packs.
The *application.properties*, *application-<valuepack>.properties*, *custom-application-<valuepack>.properties* property files have to be copied to *src/main/resources/* directory to start the application from Eclipse.

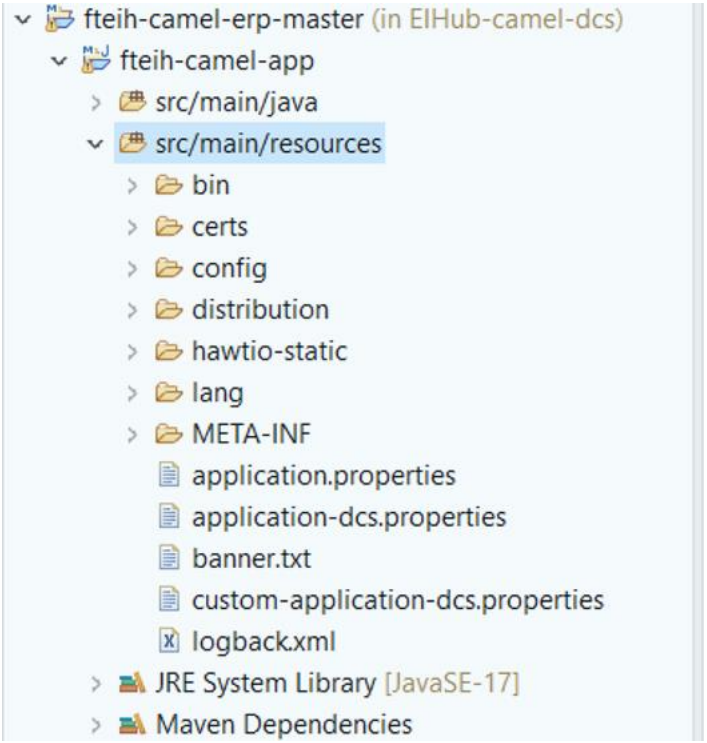

Figure 4: Camel Spring-Boot project

- Groovy files:

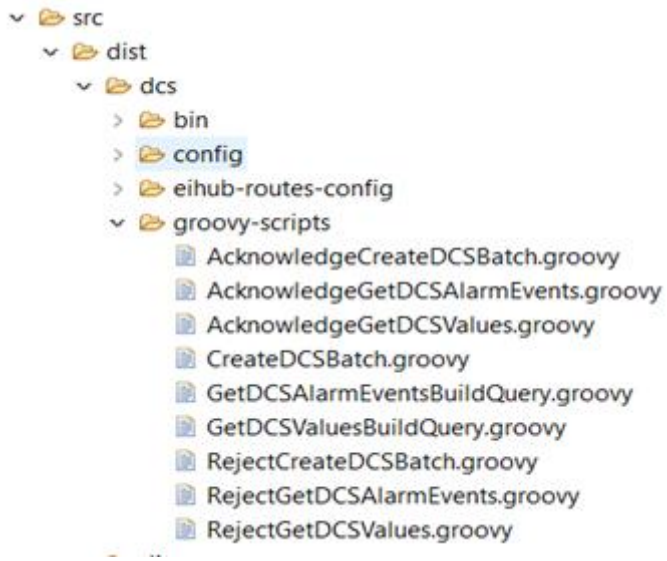

Figure 5: Groovy files in the ElHub-app project

- Route template definitions:

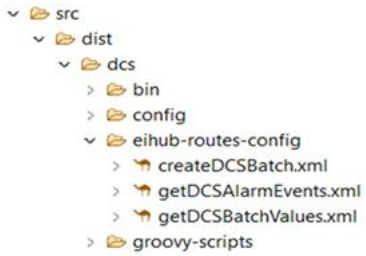

Figure 6: Route templates in the ElHub-app project

- Template Builder:
  - Locate the *dcs-route-builder.xml* file in *src/dist/dcs/config* folder and add the *templatedRoute* here to start the route.

4. On the parent project click **Run As > Maven clean install** to clean the workspace and resolve all dependencies.

5. When the setup is complete and all the properties are configured correctly, open *EIHubAppAdmin.java* and click **Run as > Java Application** to start EIHub Application from the Eclipse.

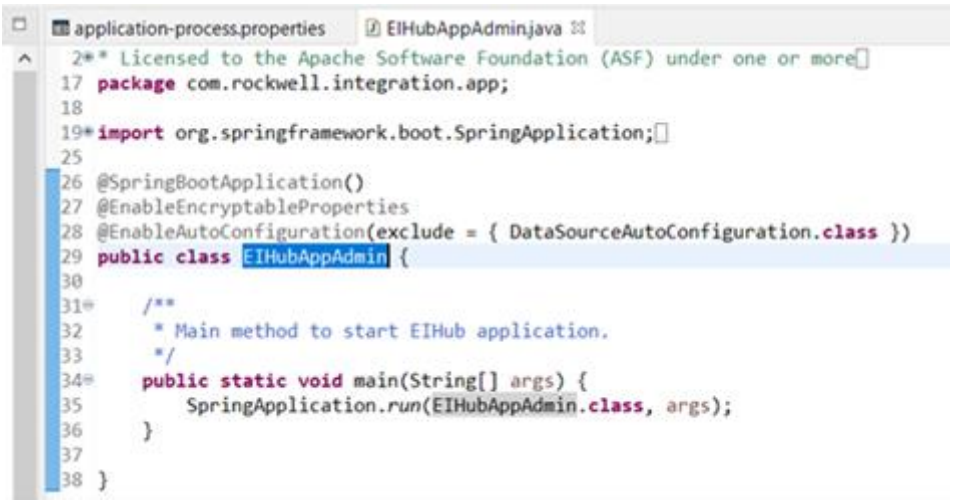

Figure 7: Springboot application class

IMPORTANT

The Camel route diagrams are not visible in Eclipse. JBoss Fuse Tooling is deprecated.

Creating Custom Routes

Custom routes can be created in **fteih-camel-app** project and the path to the created route is added to the property files. To create and configure custom routes, see Externalized Configuration for Routes (page 47).

TIP

Camel route templates created in the *src/dist/<valuepack>/eihub-routes-config* directory are automatically loaded during the startup. When route templates are added elsewhere, add the path to the application property.

```
camel.springboot.routes-include-pattern=
file:src/dist/dcs/eihub-routes-config/**/*.xml,file:src/dist/dcs/config/dcs-route-builder.xml
```

Deploying Custom Artifacts

After performing all the customizations, proceed as follows to move the changes to the installed EIHub Application:

1. Select **Run As** > **Maven clean install** on the parent project (e.g.: **EIHUB-camel**) to generate the JAR archive.
2. Copy the following JAR files generated in the respective project's target directory to *EIHub-app-<valuepack>-<FTEI version>.0xx\lib* directory:
    1. *fteih-camel-app-<FTEI version>.0xx.jar*
    2. *fteih-camel-dcs-vp-<FTEI version>.0xx.jar*

    TIP
    Copy the jars files generated from custom projects created to the installed EIHub Application's lib directory, i.e., *EIHub-app-<valuepack>-<FTEI version>.0xx\lib*.
3. Copy other updated configuration files to the respective directories:
    1. Copy the application property file (*custom-application-<valuepack>.properties*) and the route builder file (*<valuepack>-route-builder.xml)* to the *EIHub-app-<valuepack>-<FTEI version>.0xx\config* directory.
    2. Copy the route template files to the *<EIHub-app-<valuepack>-<FTEI version>.0xx>\eihub-routes-config* directory.
    3. Copy groovy script files to *<EIHub-app-<valuepack>-<FTEI version>.0xx>\groovy-scripts* directory.
    4. Copy email template files to *<EIHub-app-<valuepack>-<FTEI version>.0xx>\email-templates* directory.
4. Restart the EIHub application.

Chapter 4

Implementing DCS Adapter Integration Touchpoints

Overview of Data Transformation

By using the DCS Adapter interface, the Data Transformation can be considered as transforming the DCS Adapter B2MML request documents into DCS-specific requests and vice versa. The MES will always use JMS (ActiveMQ) as the interface technology, which means the MES will post the required B2MML message on a relevant queue or will read a B2MML message from a relevant queue.

DCS-specific requests can use a diverse set of interfacing technologies such as the following:

- RESTful service calls
- SOAP-based web service calls
- Database access using JDBC
- File-based data exchange
- Proprietary protocols on TCP/IP

EIHub provides a large set of interface components (Apache Camel components).

Following components have been used in out of the box DCS integration examples:

- RESTful service calls
- Database access using JDBC

For using other components from Apache Camel, see the Apache Camel User Manual [C1] (page 3).

Carefully select the right component for the DCS integration job. In some cases, you may need to develop a new component or develop a non-Java lightweight daemon process that implements platform-specific protocols while exposing a Camel-compatible protocol like RESTFul or JMS. In any case, planning is required for specific DCS integration to work.

The steps summarized the process for developing a PharmaSuite DCS integration touchpoint are as follows:

1. Determine the Camel components used for the specific integration touchpoint. The following should be specified:
   - ActiveMQ endpoint configuration
   - DCS-specific endpoint configuration (e.g., JDBC connectivity or HTTP(s))
2. Generate the Data Transformation logic.

   Write the logic to transform the B2MML payload into a DCS-specific payload (e.g., transform the B2MML payload into a SQL request payload or transform the B2MML payload into a JSON payload). This can be written via a Groovy script of a Java Bean class or a Java Transformer class.
3. Determine the Exception cases.

   Specify the exceptions that may occur during the route processing and implement the exception handling behavior, which will mostly result in generating a B2MML rejection response message.
4. For DCS-initiated integration touchpoints, adapt the actions in PharmaSuite as needed.

Developing a Get DCS Alarms Integration Touchpoint

This section details the tasks required to develop a Get DCS Alarms integration touchpoint The example uses integration with PlantPAx.

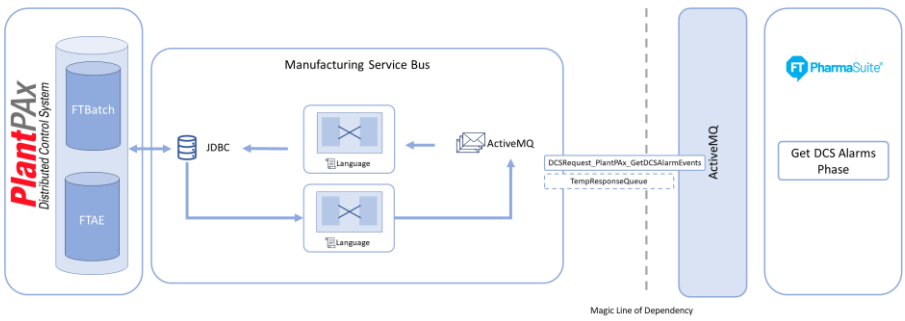

Figure 8: Get DCS alarms integration touchpoint

The specific PlantPAx connector for retrieving the alarm events is based upon database access. In essence, the PlantPAx FT Batch and FT Alarm and Events database archiving must be set up as follows:

- The FT Batch database must be extended to add specific database fields and modified stored procedures in order to have UTC-based timestamps.
- The FT Alarm and Events ConditionEvent table must be made available as a 1:1 view in the BatchHistory database.

If these specific configurations are not in place, you will need to define other SQL requests within the Groovy Data Transformation scripts.

To setup the PlantPAx Get DCS AlarmEvents route, proceed as follows:

name	value
spring.config.import	custom-application-${spring.profiles.active}.properties
camel.springboot.routes-include-pattern	file:src/dist/dcs/eihub-routes-config/**/*.xml,file:src/dist/dcs/config/dcs-route-builder.xml
camel.springboot.name	pharmasuite-plantpax-msb-context
GMPAlarm.severityLevel	0
DCS_Adapter	PlantPAx
groovyScript.path.dcs	file:src/dist/dcs/groovy-scripts/
spring.activemq.broker-url	failover://(ssl://<hostname>:<port>)?startupMaxReconnectAttempts=15
spring.activemq.user	user
spring.activemq.password	password
spring.activemq.trust-store	file:src/main/resources/certs/<truststore>.p12
spring.activemq.trust-store-password	password
getdcsvalues.numericScale	3
batchserver.url	https://<hostname>:<port>/batches
batchserver.user	user
batchserver.password	password
batchserver.authenticationMethod	Basic
trustStore.file	file:src/dist/dcs/certs/<filename>.p12
keyManagers.keyPassword	password
trustStore.password	password
spring.datasource.url	jdbc:sqlserver://<hostname>:<port>;databaseName=<DBName>;trustServerCertificate=true;
spring.datasource.username	username
spring.datasource.password	password
spring.datasource.driverClassName	com.microsoft.sqlserver.jdbc.SQLServerDriver

Figure 9: Application-specific properties

1. Define application-specific properties required for this route:

TIP
Encrypt Passwords using the RunJasyptPasswordUtil script present in *<EIHUB_Install>\installers\EIHub\Utility-Tools\encryption-tool-<FTEI version>.0xx.zip* file. Encrypted passwords in the property file are decrypted using the properties present in the *application.properties*:
`jasypt.encryptor.password==${ACTIVEMQ_ENCRYPTION_PASSWORD}`
`jasypt.encryptor.algorithm=PBEWITHHMACSHA256ANDAES_256`

2. Define the bean required for ActiveMQ integration, as shown in the following script. This bean is present in *com.rockwell.integration.app.config.JmsConfiguration* class:

```
    */
    @Bean
    public ActiveMQSslConnectionFactory activeMQSslConnectionFactory() throws Exception {
        ActiveMQSslConnectionFactory factory = new ActiveMQSslConnectionFactory(url);
        if (trustStore != null && !trustStore.isEmpty()) {
            factory.setTrustStoreType("pkcs12");
            factory.setTrustStore(new File(trustStore).toURI().toString());
            factory.setTrustStorePassword(trustStorePassword);
        }
        factory.setUserName(user);
        factory.setPassword(password);
        factory.setTrustedPackages(Arrays.asList(trustedPackages.split(",")));
        return factory;
    }
}
```

Figure 10: Bean for ActiveMQ integration

Notice the use of the application property *spring.activemq.brokerURL* as URL. You also may need to set the pre-fetch limit for the consumers in order to have efficiently performing concurrent consumers. For details, see the relevant the Apache ActiveMQ Getting Started Document [C2] (page 3).

3. Define the bean required for database access in the Camel route file as shown in the following script. This bean is present in *com.rockwell.integration.app.config.DcsConfiguration* class:

```
 */
@Bean
public DataSource getDataSource() {
    return DataSourceBuilder.create().driverClassName(this.dbDriverClassName).url(this.dbUrl)
            .username(this.dbUsername).password(this.dbPassword).build();
}
```

Figure 11: Bean required for database access

Notice the use of the application properties relevant to database access from the *application-dcs.properties* file.

4. Define the route Templates and configure the route starting with the ActiveMQ endpoint from which you need to read the *GetDCSAlarmEvent* XML message as shown in the following script:

```
<routeTemplates xmlns="http://camel.apache.org/schema/spring">
  <routeTemplate id="getDCSAlarmEventsTemplate">
     <route id="getDCSAlarmEvents">
     <description>Read getDCSAlarmEvents BML message from queue,
     transform into a SQL request, execute the SQL retrieval
     and transform the resultset response into the BML response</description>
     <from id="getDCSAlarmEventsMessageFromQueue"
     uri="activemq:queue:DCSRequest_{{DCS_Adapter}}_GetDCSAlarmEvents?exchangePattern=InOut"/>
```

Figure 12: Configuring the route

Notice the use of the *DCS_Adapter* application property.

5. Add the transformation of the incoming payload to a SQL request. The processing is done via the *GetDCSAlarmEventsBuildQuery.groovy* script.

```
<to id="transformToDCSAlarmEventsQuery" pattern="InOut"
uri="language:groovy:resourceUri?script={{groovyScript.path.dcs}}GetDCSAlarmEventsBuildQuery.groovy"/>
```

Figure 13: Adding the transformation of the incoming payload to a SQL request

The Groovy script contains the logic to transform the XML payload into a SQL request:

```
import groovy.xml.*;
import org.apache.camel.*;


DATETIMEFORMAT = '''yyyy-MM-dd'T'HH:mm:ss.SSSXXX'''
SIMPLEDATETIMEFORMAT = '''yyyy-MM-dd HH:mm:ss.SSS'''
/*

Important remark:
The implementation requires a simple 1:1 view of the FT Alarms & Events table ConditionEvent
in the BatchHistory DB.
*/

def convertToUTCString(cal) {


   sdf = new java.text.SimpleDateFormat(SIMPLEDATETIMEFORMAT);
   sdf.setTimeZone(TimeZone.getTimeZone("UTC"));
   utcTimeString = sdf.format(cal.getTime());
   return utcTimeString;
}

def selectFields = '''SELECT CE.EventTimeStamp, CE.SourceName, CE.Message,
CE.ConditionName, CE.AlarmClass, CE.InputValue, CE.LimitValue, CE.Active,
CE.EventCategory, CE.Severity, CE.Priority ''';
def ftaeTable = '''FTAE_ConditionEvent CE ''';
def baseFrom = '''FROM ''';
def baseWhere = '''WHERE CE.Severity> :?severity AND ISNULL(CE.Active,0) = 1''';
def xmlDoc = new XmlSlurper(false,false).parseText(request.getBody().toString());
def sender = xmlDoc.ApplicationArea.Sender.text();
def receiver = xmlDoc.ApplicationArea.Receiver.text();
def batchID = xmlDoc.DataArea.BatchID.text();
def equipmentID = xmlDoc.DataArea.EquipmentID.text();
def startTime =  xmlDoc.DataArea.StartTime.text();
def endTime =  xmlDoc.DataArea.EndTime.text();
def severity =
camelContext.resolvePropertyPlaceholders("{{GMPAlarm.severityLevel}}").toString();
// setting base flow vars
//

exchange.setProperty("sender", sender)
exchange.setProperty("receiver", receiver)
exchange.setProperty("batchID", batchID)
exchange.setProperty("equipmentID",equipmentID)
def substMap = [:]
def clientTimezone = null;
substMap['severity']=severity;
def whereOptions='''''';
if (startTime ) { // means it is not null and not empty
```

```
   baseWhere += '''AND CE.EventTimeStamp >= :?startTime ''';
    aCal = javax.xml.bind.DatatypeConverter.parseDateTime(startTime);
    clientTimezone = aCal.getTimeZone()
    utcString = convertToUTCString(aCal)
    substMap['startTime'] = java.sql.Timestamp.valueOf(utcString);
}

if (endTime ) { // means it is not null and not empty
   baseWhere += '''AND CE.EventTimeStamp <= :?endTime ''';
      aCal = javax.xml.bind.DatatypeConverter.parseDateTime(endTime);
      clientTimezone = aCal.getTimeZone()
      utcString = convertToUTCString(aCal)
      substMap['endTime'] = java.sql.Timestamp.valueOf(utcString);
}

exchange.setProperty("clientTimezone", clientTimezone)
if (batchID) {  // means it is not null and not empty
   baseFrom += ''' BHUnit INNER JOIN BHBatch ON BHUnit.uniqueid = BHBatch.uniqueid AND
BHBatch.batchid = :?batchId ''';
   baseFrom += 'INNER JOIN FTAE_ConditionEvent CE ON  CE.EventTimeStamp >=
BHUnit.starttime_gmt AND CE.EventTimeStamp <= BHUnit.endtime_gmt ';
   substMap['batchId']=batchID;
   if (equipmentID) {
      baseFrom += ''' AND BHUnit.unitname = :?equipmentId ''';
         substMap['equipmentId']=equipmentID;
   }
} else { // no batch
   if (equipmentID) { // we have equipment
 //     baseFrom +=''' BHUnit INNER JOIN FTAE_ConditionEvent CE ON  CE.EventTimeStamp >=
BHUnit.starttime AND CE.EventTimeStamp <= BHUnit.endtime ''';
 //     baseFrom += ''' AND BHUnit.unitname =equipmentId ''';
        baseFrom +=''' FTAE_ConditionEvent CE ''';
       substMap['equipmentId']=equipmentID;
   } else {
     baseFrom += ftaeTable;
   }
}

def controlModuleMap = [:]
def ctr = 0;
def baseCtrModuleName='actualEquipmentId';
substMap['equipmentId1']="%"+equipmentID+"%";
def tpl1 = ''' CE.SourceName like ''';
def tpl2 =  ''' CE.SourceName like :?''';
xmlDoc.DataArea.RecipeElement.ActualEquipmentID.each { elt ->
   substMap[baseCtrModuleName + ctr.toString()] = "%"+elt.text()+"%";
  if (ctr == 0) {
      baseWhere += ''' AND ( ''';
  } else { // greater than
       baseWhere += ''' OR ''';
  }

  if (equipmentID) {
     substMap[baseCtrModuleName + ctr.toString()] ="%"+equipmentID +"%"+elt.text()+"%";
     baseWhere += tpl1+''':?'''+(baseCtrModuleName + ctr.toString());
  } else {
       baseWhere += tpl2 +(baseCtrModuleName + ctr.toString());
  }

  ctr++;
}
```

```
if (ctr > 0) {
   baseWhere += ''')''';
}

if (ctr == 0 && equipmentID) { // no control modules
      baseWhere += ''' AND CE.SourceName like :?equipmentId1 ''';
}

def sqlString = selectFields + baseFrom + baseWhere;
response.setHeader("CamelSqlQuery", sqlString)
return substMap
```

TIP

It is important to understand how the query is generated. GMP Critical alarms are currently viewed as alarm events that have a severity of whatever the property is configured. It is up to the developer or integrator to define the right queries reflecting the PlantPAx setup and to adapt the Groovy script accordingly.

6. Issue the SQL request to the database. By default, the placeholder is used to define the SQL statement. In this case, this will be the payload generated by the previous Groovy script executed in the route.

```
<to id="retrieveDSCAlarmEventsBySQL" pattern="InOut"
uri="sql:select 1?exchangePattern=InOut&amp;dataSource=#SQLServerPlantPAXBatchHist"/>
```

Figure 14: SQL request to the database

7. Response from executed query is processed by the groovy file, *AcknowledgeGetDCSAlarmEvents.groovy*.

```
<to id="transformDSCAlarmEventsResultSet" pattern="InOut"
uri="language:groovy:resourceUri?
script={{groovyScript.path.dcs}}AcknowledgeGetDCSAlarmEvents.groovy"/>
```

Figure 15: Transform the ResultSet

```
import groovy.xml.*;
import org.apache.camel.*;
import java.util.*;


DATETIMEFORMAT = '''yyyy-MM-dd'T'HH:mm:ss.SSSXXX'''
SIMPLEDATETIMEFORMAT = '''yyyy-MM-dd HH:mm:ss.SSS'''


LOGGER = org.apache.logging.log4j.LogManager.getLogger("Rockwell")
LOGGER.debug("Entering AcknowledgeGetDCSAlarmEvents");


def generateAlarmValue(record) {
   def pattern = '''Category:$eventCategory;Severity:$severity;Priority:$priority;
Condition:$condition;
InputValue:$inputValue;LimitValue:$limitValue;$message'''
   def recordMap = [:]
   recordMap['eventCategory']=record.EventCategory
```

```
   recordMap['severity']=record.Severity
   recordMap['priority']=record.Priority
   recordMap['condition']=record.ConditionName
   recordMap['inputValue']=record.InputValue
   recordMap['limitValue']=record.LimitValue
   recordMap['message']=record.Message
   def alarmValueEngine = new groovy.text.SimpleTemplateEngine();
   def alarmTpl = alarmValueEngine.createTemplate(pattern).make(recordMap);
   return alarmTpl.toString()


}


def generateEquipmentId(equipmentId, record) {


   if (equipmentId){
     return equipmentId + "/" + record.SourceName

   }
   return record.SourceName
}


def generateDateTime(record, timezone) {
 cal = new GregorianCalendar();
 cal.setTimeZone(TimeZone.getTimeZone('UTC'))
 cal.setTime(record.EventTimeStamp);


return cal.format(DATETIMEFORMAT);
}


def convertToUTCString(cal) {


   sdf = new java.text.SimpleDateFormat(SIMPLEDATETIMEFORMAT);
   sdf.setTimeZone(TimeZone.getTimeZone("UTC"));
   utcTimeString = sdf.format(cal.getTime());
   return utcTimeString;


}


def convertNumeric(stringValue) {
   bdValue = new BigDecimal(stringValue);
   roundedBDValue = new BigDecimal(bdValue).setScale
(camelContext.resolvePropertyPlaceholders
("{{getdcsvalues.numericScale}}").toInteger(),
       BigDecimal.ROUND_HALF_UP);
   return roundedBDValue.toString();
}


def queryResult = request.getBody();
def receiver = exchange.getProperty("sender")
def sender = exchange.getProperty("receiver")
```

```
def writer = new StringWriter()
def xml = new MarkupBuilder(writer)
xml.ShowDCSAlarmEvents('xsi:schemaLocation':'http://www.rockwell.com/mes/dcs/ifc
../../../main/xsd/mes-dcs-interface.xsd',
                       'xmlns':'http://www.rockwell.com/mes/dcs/ifc',
                       'xmlns:xsi':'http://www.w3.org/2001/XMLSchema-instance',
 'schemeVersionID':'1.0') {

    ApplicationArea {
        Sender(sender)
        Receiver(receiver)
    }
    DataArea {
        ResponseCriteria(actionCode:"Accepted")
        Events {
           queryResult.each  {record ->
              AlarmEvent {


                 TimeStamp(generateDateTime(record,
exchange.getProperty('clientTimezone')))
                 Value(generateAlarmValue(record))
                 EquipmentID(generateEquipmentId(exchange.getProperty('equipmentID'),
record))
                 MessageText(record.Message)
              }
            }

        }


     }

  }
return writer.toString()
```

8. Add exception management in order to generate a reject message. The generation of the reject message is done via the *RejectGetDCSAlarmEvents.groovy* script .

```
<onException id="getDCSAlarmEvents_multipleCauseException">
    <exception>java.lang.Exception</exception>
    <handled>
        <constant>true</constant>
    </handled>
    <to id="generateDCSAlarmEventsRejectResponse"
        pattern="InOut" uri="language:groovy:resourceUri?
        script={{groovyScript.path.dcs}}RejectGetDCSAlarmEvents.groovy"/>
    <log message="${body}"/>
</onException>
```

Figure 16: Generating the reject message

If an exception is thrown during the life of the route's exchange, a reject message will be generated via the Groovy script.

```
import groovy.xml.*;
import org.apache.camel.*;
import java.util.*;

def receiver = exchange.getProperty("sender") != null ? exchange.getProperty("sender")
: "UNKNOWN"

def sender = exchange.getProperty("receiver") != null ? exchange.getProperty("receiver")
: "UNKNOWN"

def cause = exchange.getProperty(org.apache.camel.Exchange.EXCEPTION_CAUGHT,
Exception.class);

def excClassName = cause.getClass().getName()
def errorMessage = excClassName + ': Please check log files on MessageBroker
for more details'

def writer = new StringWriter()

def xml = new MarkupBuilder(writer)
xml.ShowDCSAlarmEvents('xsi:schemaLocation':'http://www.rockwell.com/mes/dcs/ifc
../../../main/xsd/mes-dcs-interface.xsd',
                   'xmlns':'http://www.rockwell.com/mes/dcs/ifc',
                   'xmlns:xsi':'http://www.w3.org/2001/XMLSchema-instance',
'schemeVersionID':'1.0')
    { ApplicationArea {
        Sender(sender)
        Receiver(receiver)
    }
    DataArea {
        ResponseCriteria(actionCode:'Rejected', errorMessage)
    }
}
return writer.toString()
```

Developing a Get DCS Batch Values Integration Touchpoint

This section details the tasks required to develop a Get DCS Batch Values integration touchpoint. The example uses integration with PlantPAx.

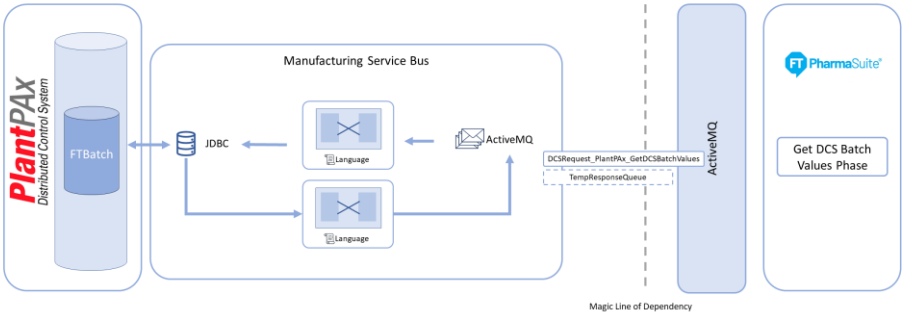

Figure 17: Get DCS batch values integration touchpoint

The specific PlantPAx connector for retrieving the Batch Values is based upon database access. In essence, the PlantPAx FT Batch and FT Alarm and Events database archiving must be set up as follows:

- The FT Batch database must be extended to add specific database fields and modified stored procedures in order to have UTC-based timestamps.
- The FT Alarm and Events ConditionEvent table must be made available as a 1:1 view in the BatchHistory database.

If these specific configurations are not in place, you will need to define other SQL requests within the Groovy Data Transformation scripts.

The following steps outline the setup of the PlantPAx Get DCS Batch Values route.

name	value
spring.config.import	custom-application-${spring.profiles.active}.properties
camel.springboot.routes-include-pattern	file:src/dist/dcs/eihub-routes-config/**/*.xml,file:src/dist/dcs/config/dcs-route-builder.xml
camel.springboot.name	pharmasuite-plantpax-msb-context
GMPAlarm.severityLevel	0
DCS_Adapter	PlantPAx
groovyScript.path.dcs	file:src/dist/dcs/groovy-scripts/
spring.activemq.broker-url	failover://(ssl://<hostname>:<port>)?startupMaxReconnectAttempts=15
spring.activemq.user	user
spring.activemq.password	password
spring.activemq.trust-store	file:src/main/resources/certs/<truststore>.p12
spring.activemq.trust-store-password	password
getdcsvalues.numericScale	3
batchserver.url	https://<hostname>:<port>/batches
batchserver.user	user
batchserver.password	password
batchserver.authenticationMethod	Basic
trustStore.file	file:src/dist/dcs/certs/<filename>.p12
keyManagers.keyPassword	password
trustStore.password	password
spring.datasource.url	jdbc:sqlserver://<hostname>:<port>;databaseName=<DBName>;trustServerCertificate=true;
spring.datasource.username	username
spring.datasource.password	password
spring.datasource.driverClassName	com.microsoft.sqlserver.jdbc.SQLServerDriver

Figure 18: Application-specific properties

1. Define application-specific properties required for this route:
2. Define the bean required for ActiveMQ integration with the Camel Context as shown in the following script:

```
    @Bean
    public ActiveMQSslConnectionFactory activeMQSslConnectionFactory() throws Exception {
        ActiveMQSslConnectionFactory factory = new ActiveMQSslConnectionFactory(url);
        if (trustStore != null && !trustStore.isEmpty()) {
            factory.setTrustStoreType("pkcs12");
            factory.setTrustStore(new File(trustStore).toURI().toString());
            factory.setTrustStorePassword(trustStorePassword);
        }
        factory.setUserName(user);
        factory.setPassword(password);
        factory.setTrustedPackages(Arrays.asList(trustedPackages.split(",")));
        return factory;
    }
}
```

Figure 19: Bean for ActiveMQ integration

Notice the use of the application property *activemq.brokerURL* as URL.
You also may need to set the pre-fetch limit for the consumers in order to have efficiently performing concurrent consumers. For details, see the Apache ActiveMQ Getting Started Document [C2] (page 3).

3. Define the bean required for database access in the Camel route, as shown in the following script:

```
 */
@Bean
public DataSource getDataSource() {
    return DataSourceBuilder.create().driverClassName(this.dbDriverClassName).url(this.dbUrl)
            .username(this.dbUsername).password(this.dbPassword).build();
}
```

Figure 20: Bean required for database access

Notice the use of the application properties relevant to database access.

4. Define the route templates and configure the route starting with the ActiveMQ endpoint from which you need to read the *GetDCSBatchValues* XML message as shown in the following script:

```
<routeTemplates xmlns="http://camel.apache.org/schema/spring">
  <routeTemplate id="getDCSAlarmEventsTemplate">
    <route id="getDCSAlarmEvents">
    <description>Read getDCSAlarmEvents BML message from queue,
    transform into a SQL request, execute the SQL retrieval
    and transform the resultset response into the BML response</description>
    <from id="getDCSAlarmEventsMessageFromQueue"
    uri="activemq:queue:DCSRequest_{{DCS_Adapter}}_GetDCSAlarmEvents?exchangePattern=InOut"/>
```

Figure 21: Configuring the route

Notice the use of *DCS_Adapter* application property.

5. Add the transformation of the incoming payload to a SQL request. The processing is done via the *GetDCSValuesBuildQuery.groovy* script.

```
<to id="transformToDCSAlarmEventsQuery" pattern="InOut"
uri="language:groovy:resourceUri?script={{groovyScript.path.dcs}}GetDCSAlarmEventsBuildQuery.groovy"/>
```

Figure 22: Adding the transformation of the incoming payload to a SQL request

The Groovy script contains the logic to transform the XML payload into a SQL request:

```
import groovy.xml.*;
import org.apache.camel.*;

DATETIMEFORMAT = '''yyyy-MM-dd'T'HH:mm:ss.SSSXXX'''
SIMPLEDATETIMEFORMAT = '''yyyy-MM-dd HH:mm:ss.SSS'''
/*
Important remark:
The implementation requires a simple 1:1 view of the FT Alarms & Events table ConditionEvent
in the BatchHistory DB.
*/
def convertToUTCString(cal) {


   sdf = new java.text.SimpleDateFormat(SIMPLEDATETIMEFORMAT);
   sdf.setTimeZone(TimeZone.getTimeZone("UTC"));
   utcTimeString = sdf.format(cal.getTime());
   return utcTimeString;
```

```
}
def selectFields = '''SELECT CE.EventTimeStamp, CE.SourceName, CE.Message,
CE.ConditionName, CE.AlarmClass, CE.InputValue, CE.LimitValue, CE.Active,
CE.EventCategory, CE.Severity, CE.Priority ''';
def ftaeTable = '''FTAE_ConditionEvent CE ''';
def baseFrom = '''FROM ''';
def baseWhere = '''WHERE CE.Severity> :?severity AND ISNULL(CE.Active,0) = 1''';
def xmlDoc = new XmlSlurper(false,false).parseText(request.getBody().toString());
def sender = xmlDoc.ApplicationArea.Sender.text();
def receiver = xmlDoc.ApplicationArea.Receiver.text();
def batchID = xmlDoc.DataArea.BatchID.text();
def equipmentID = xmlDoc.DataArea.EquipmentID.text();
def startTime =  xmlDoc.DataArea.StartTime.text();
def endTime =  xmlDoc.DataArea.EndTime.text();
def severity =
camelContext.resolvePropertyPlaceholders("{{GMPAlarm.severityLevel}}").toString();
// setting base flow vars
//
exchange.setProperty("sender", sender)
exchange.setProperty("receiver", receiver)
exchange.setProperty("batchID", batchID)
exchange.setProperty("equipmentID",equipmentID)
def substMap = [:]
def clientTimezone = null;
substMap['severity']=severity;
def whereOptions='''''';
if (startTime ) { // means it is not null and not empty
   baseWhere += '''AND CE.EventTimeStamp >= :?startTime ''';
    aCal = javax.xml.bind.DatatypeConverter.parseDateTime(startTime);
    clientTimezone = aCal.getTimeZone()
    utcString = convertToUTCString(aCal)
    substMap['startTime'] = java.sql.Timestamp.valueOf(utcString);
}
if (endTime ) { // means it is not null and not empty
   baseWhere += '''AND CE.EventTimeStamp <= :?endTime ''';
       aCal = javax.xml.bind.DatatypeConverter.parseDateTime(endTime);
       clientTimezone = aCal.getTimeZone()
       utcString = convertToUTCString(aCal)
       substMap['endTime'] = java.sql.Timestamp.valueOf(utcString);
}
exchange.setProperty("clientTimezone", clientTimezone)
if (batchID) {  // means it is not null and not empty
   baseFrom += ''' BHUnit INNER JOIN BHBatch ON BHUnit.uniqueid = BHBatch.uniqueid AND
BHBatch.batchid = :?batchId ''';
   baseFrom += 'INNER JOIN FTAE_ConditionEvent CE ON  CE.EventTimeStamp >=
BHUnit.starttime_gmt AND CE.EventTimeStamp <= BHUnit.endtime_gmt ';
   substMap['batchId']=batchID;
   if (equipmentID) {
      baseFrom += ''' AND BHUnit.unitname = :?equipmentId ''';
         substMap['equipmentId']=equipmentID;
   }
} else { // no batch
   if (equipmentID) { // we have equipment
 //     baseFrom +=''' BHUnit INNER JOIN FTAE_ConditionEvent CE ON  CE.EventTimeStamp >=
BHUnit.starttime AND CE.EventTimeStamp <= BHUnit.endtime ''';
 //     baseFrom += ''' AND BHUnit.unitname =equipmentId ''';
        baseFrom +=''' FTAE_ConditionEvent CE ''';
       substMap['equipmentId']=equipmentID;
   } else {
     baseFrom += ftaeTable;
   }
}
```

```
def controlModuleMap = [:]
def ctr = 0;
def baseCtrModuleName='actualEquipmentId';
substMap['equipmentId1']="%"+equipmentID+"%";
def tpl1 = ''' CE.SourceName like ''';
def tpl2 =  ''' CE.SourceName like :?''';
xmlDoc.DataArea.RecipeElement.ActualEquipmentID.each { elt ->
   substMap[baseCtrModuleName + ctr.toString()] = "%"+elt.text()+"%";
  if (ctr == 0) {
      baseWhere += ''' AND ( ''';
  } else { // greater than
       baseWhere += ''' OR ''';
  }
  if (equipmentID) {
     substMap[baseCtrModuleName + ctr.toString()] ="%"+equipmentID +"%"+elt.text()+"%";
     baseWhere += tpl1+''':?'''+(baseCtrModuleName + ctr.toString());
  } else {
       baseWhere += tpl2 +(baseCtrModuleName + ctr.toString());
  }
  ctr++;
}
if (ctr > 0) {
   baseWhere += ''')''';
}
if (ctr == 0 && equipmentID) { // no control modules
     baseWhere += ''' AND CE.SourceName like :?equipmentId1 ''';
}
def sqlString = selectFields + baseFrom + baseWhere;
response.setHeader("CamelSqlQuery", sqlString)
return substMap
```

TIP
It is important to understand how the query is generated. GMP Critical alarms are currently viewed as alarm events that have a severity of whatever the property is configured. It is up to the developer or integrator to define the right queries reflecting the PlantPAx setup and to adapt the Groovy script accordingly.

6. Issue the SQL request to the database. By default, the placeholder is used to define the SQL statement. In this case, this will be the payload generated by the previous Groovy script executed in the route.

```
<to id="retrieveDSCAlarmEventsBySQL" pattern="InOut"
uri="sql:select 1?exchangePattern=InOut&amp;dataSource=#SQLServerPlantPAXBatchHist"/>
```

Figure 23: SQL request to the database

7. Response from executed query is processed by the groovy file, *AcknowldgeGetDCSValues.groovy*.

```
<to id="transformDSCAlarmEventsResultSet" pattern="InOut"
uri="language:groovy:resourceUri?
script={{groovyScript.path.dcs}}AcknowledgeGetDCSAlarmEvents.groovy"/>
```

Figure 24: Transform the ResultSet

```
import groovy.xml.*;
import org.apache.camel.*;

def convertNumeric(stringValue) {
   bdValue = new BigDecimal(stringValue);
   roundedBDValue = new
BigDecimal(bdValue).setScale(camelContext.resolvePropertyPlaceholders("{{getdcsvalues.
numericscale}}").toInteger(),
         BigDecimal.ROUND_HALF_UP);
   return roundedBDValue.toString();
}

def queryResult = request.getBody();
def receiver = exchange.getProperty("sender")
def sender = exchange.getProperty("receiver")

// payload is a list of Maps i.e. List<Map<String, Object>>
queryResult.each  {record ->
   record['Recipe'] = record['Recipe'].subSequence(record['Recipe'].indexOf(":") + 1,
record['Recipe'].length())

}
def writer = new StringWriter()
def xml = new MarkupBuilder(writer)
def  tpMap = exchange.getProperty('reportParamTypeMap')
xml.ShowDCSBatchValues('xsi:schemaLocation':'http://www.rockwell.com/mes/dcs/ifc
../../../main/xsd/mes-dcs-interface.xsd',
'xmlns':'http://www.rockwell.com/mes/dcs/ifc',
      'xmlns:xsi':'http://www.w3.org/2001/XMLSchema-instance', 'schemeVersionID':'1.0')
{
      ApplicationArea {
            Sender(sender)
            Receiver(receiver)
      }
      DataArea {
            ResponseCriteria(actionCode:'Accepted')
            ControlRecipe {
               queryResult.each  {record ->
               RecipeElement{
                     ID(record['Recipe'])
                     ParameterID(record['Descript'])

                     if (tpMap.get([record['Recipe'], record['Descript']]) == 'string') {
                           ValueString(record['PValue']);
                     } else if  (tpMap.get([record['Recipe'], record['Descript']]) ==
'boolean') {
                           ValueBoolean(record['PValue']);
                     }  else if  (tpMap.get([record['Recipe'], record['Descript']]) ==
'integer') {
                           ValueInteger(record['PValue']);
                     }  else if  (tpMap.get([record['Recipe'], record['Descript']]) ==
'decimal') {
                           ValueNumeric(convertNumeric(record['PValue']));
                     }  else if  (tpMap.get([record['Recipe'], record['Descript']]) ==
'dateTime') {
                           ValueDatetime(record['PValue']);
                     }
                    }
        }
        }
        }
        }
return writer.toString()
```

8. Add exception management in order to generate a reject message. The generation of the reject message is done via the *RejectGetDCSValues.groovy* script.

```
<onException id="getDCSAlarmEvents_multipleCauseException">
    <exception>java.lang.Exception</exception>
    <handled>
        <constant>true</constant>
    </handled>
    <to id="generateDCSAlarmEventsRejectResponse"
        pattern="InOut" uri="language:groovy:resourceUri?
        script={{groovyScript.path.dcs}}RejectGetDCSAlarmEvents.groovy"/>
    <log message="${body}"/>
</onException>
```

Figure 25: Generating the reject message

If an exception is thrown during the life of the route's exchange, a reject message will be generated via the Groovy script.

```
import groovy.xml.*;
import org.apache.camel.*;

//
// generate a reject message due to exceptions thrown during route processing
// within this reference implementation it is a catch all exception concept

def receiver = exchange.getProperty("sender") != null ? exchange.getProperty("sender")
: "UNKNOWN"

def sender = exchange.getProperty("receiver") != null ? exchange.getProperty("receiver")
: "UNKNOWN"

def cause = exchange.getProperty(org.apache.camel.Exchange.EXCEPTION_CAUGHT,
Exception.class);

def excClassName = cause.getClass().getName()
def errorMessage = excClassName + ': Please check log files on MessageBroker
for more details'

def writer = new StringWriter()
def xml = new MarkupBuilder(writer)
def  tpMap = exchange.getProperty('reportParamTypeMap')
xml.ShowDCSBatchValues('xsi:schemaLocation':'http://www.rockwell.com/mes/dcs/ifc
../../../main/xsd/mes-dcs-interface.xsd',
'xmlns':'http://www.rockwell.com/mes/dcs/ifc',
      'xmlns:xsi':'http://www.w3.org/2001/XMLSchema-instance', 'schemeVersionID':'1.0')
{
      ApplicationArea {
            Sender(sender)
            Receiver(receiver)
      }
      DataArea {
            ResponseCriteria(actionCode:'Rejected',
            errorMessage)
        }
      }
return writer.toString()
```

Developing a Create DCS Batch Touchpoint

To create DCS Batch touchpoint integration, proceed as follows:

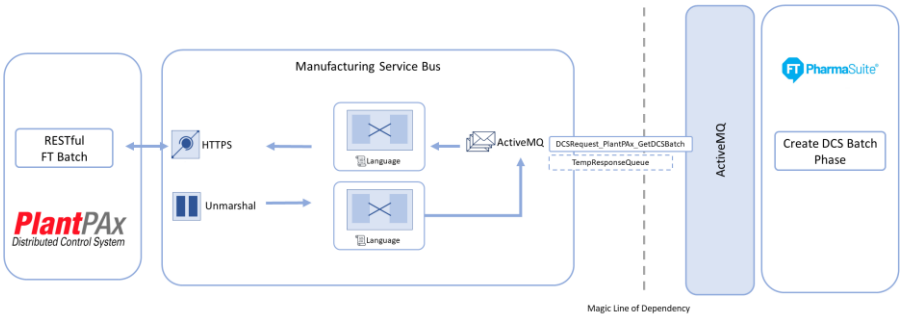

Figure 26: Create DCS batch touchpoint

TIP
It is assumed that the RESTful FT Batch server process is used and operational. The following steps outline the setup of the PlantPAx Create DCS Batch route.

1. Define application-specific properties required for this route by proceeding as follows:

name	value
spring.config.import	custom-application-${spring.profiles.active}.properties
camel.springboot.routes-include-pattern	file:src/dist/dcs/eihub-routes-config/**/*.xml,file:src/dist/dcs/config/dcs-route-builder.xml
camel.springboot.name	pharmasuite-plantpax-msb-context
GMPAlarm.severityLevel	0
DCS_Adapter	PlantPAx
groovyScript.path.dcs	file:src/dist/dcs/groovy-scripts/
spring.activemq.broker-url	failover://(ssl://<hostname>:<port>)?startupMaxReconnectAttempts=15
spring.activemq.user	user
spring.activemq.password	password
spring.activemq.trust-store	file:src/main/resources/certs/<truststore>.p12
spring.activemq.trust-store-password	password
getdcsvalues.numericScale	3
batchserver.url	https://<hostname>:<port>/batches
batchserver.user	user
batchserver.password	password
batchserver.authenticationMethod	Basic
trustStore.file	file:src/dist/dcs/certs/<filename>.p12
keyManagers.keyPassword	password
trustStore.password	password
spring.datasource.url	jdbc:sqlserver://<hostname>:<port>;databaseName=<DBName>;trustServerCertificate=true;
spring.datasource.username	username
spring.datasource.password	password
spring.datasource.driverClassName	com.microsoft.sqlserver.jdbc.SQLServerDriver

Figure 27: Application-specific properties (DCS batch touchpoint)

In the *custom-application-dcs.properties* file, add the path to truststore in the *trustStore.file* property for the Batchserver SSL configuration.

The batchserver.url property points to the RESTful web service implementing the access to the PlantPAx FT Batch system. The FT Batch RESTful interface is only accessible via SSL, so you need to set up a trustworthy keystore that will store the self-signed certificate of this server access point.

2. In order to obtain the certificate, perform the following steps:

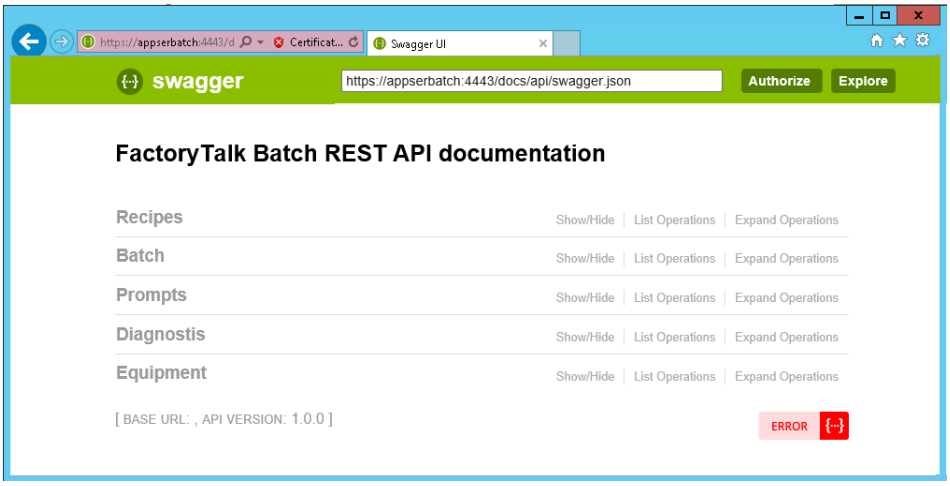

Figure 28: FactoryTalk Batch documentation

1. Download the certificate. For example, using browser, navigate to the server URL: https://appserbatch:4443/docs/api/index.html?baseUrl=appserbatch#

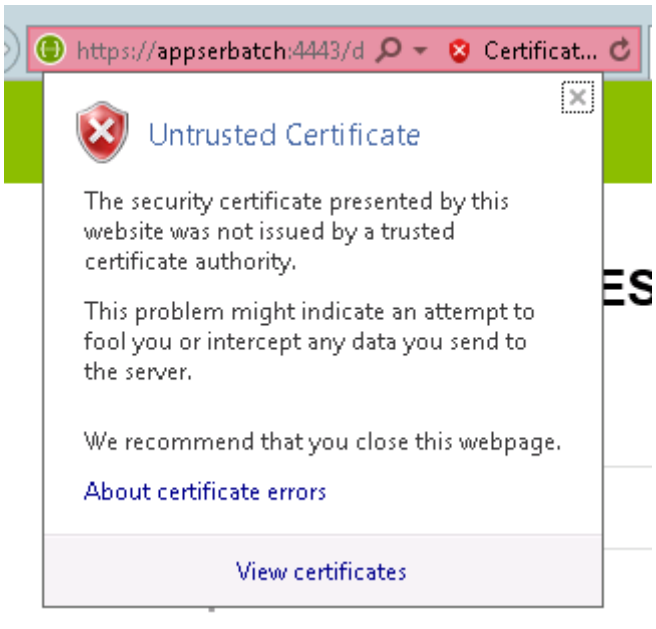

Figure 29: Certificate error

2. Click on the certificate error icon in the URL navigation field

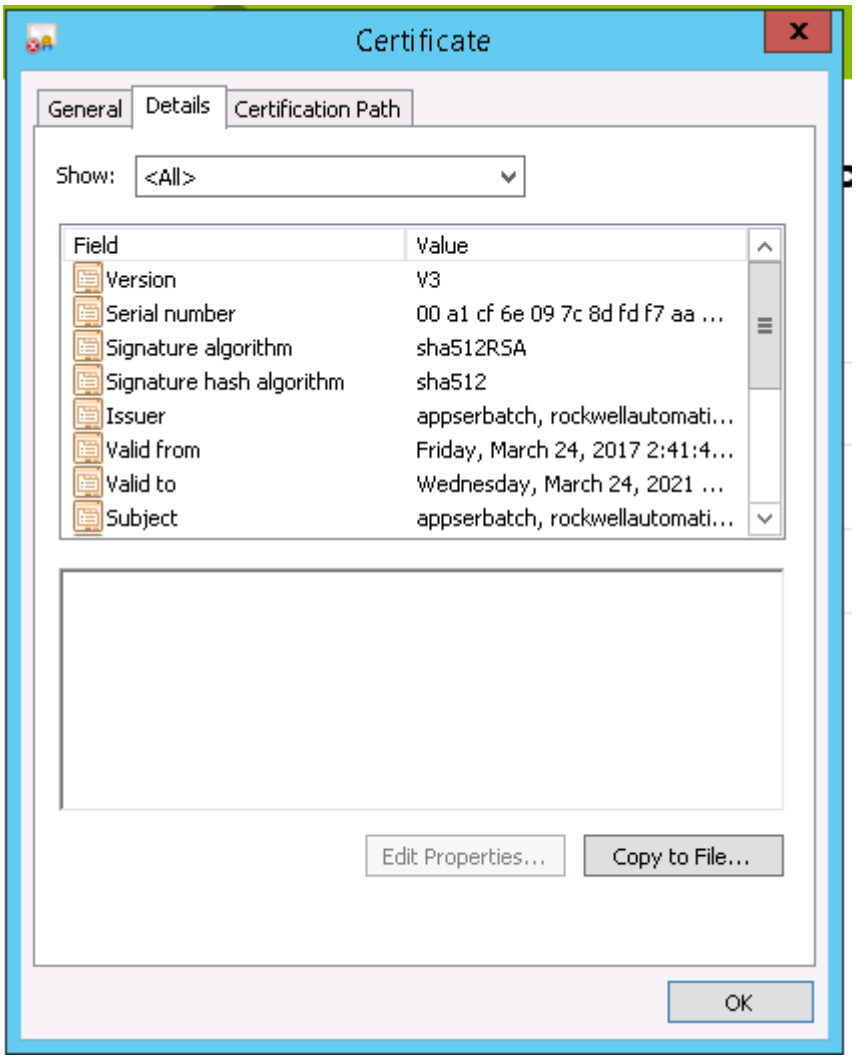

Figure 30: Details tab

3. Click **View certificates** to display the **Certificate** dialog and then select the **Details** tab.

Figure 31: Certificate Export Wizard welcome screen

4. Click **[Copy to File...]** to launch the **Certificate Export Wizard** and then click **[Next]**.

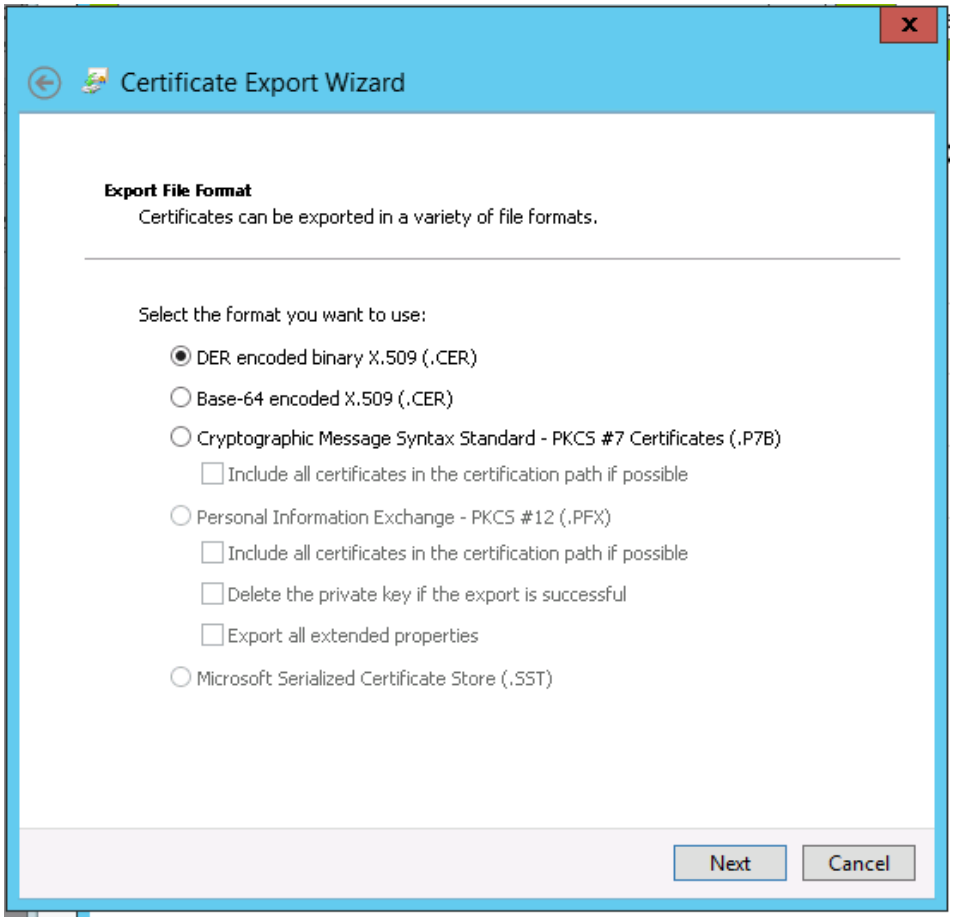

Figure 32: Export File Format screen

5. Select the **DER encoded binary X.509 (.CER)** option and click **[Next]**.

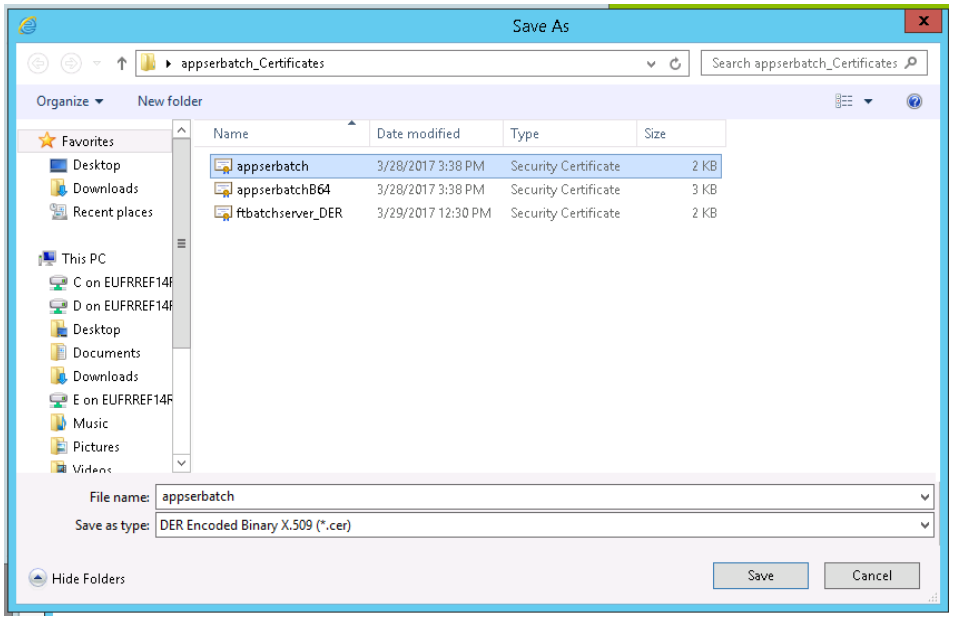

Figure 33: Export file name and location

6. Define the file name and location. Click **[Save]**.

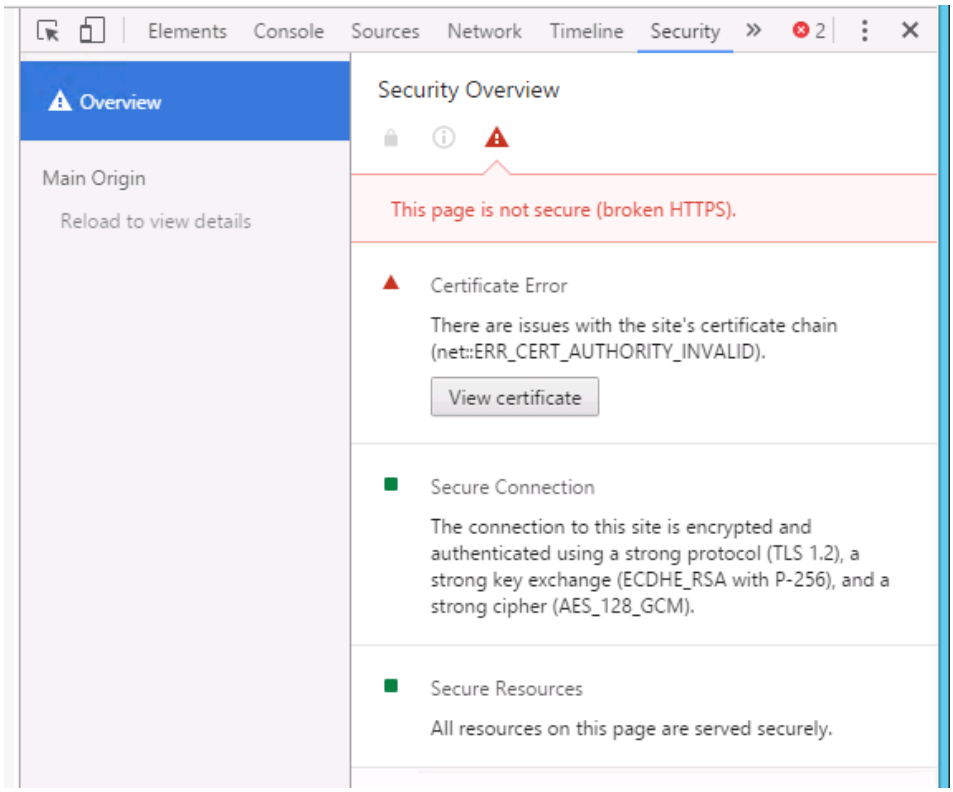

Figure 34: Developer options

7. Access the certificate. For example, using Chrome, use the Developer options to get the certificate.
8. Use the Java keytool to create a local keystore and insert this certificate.

```
keytool -import -v -trustcacerts -alias <server-alias> -file <certificate location>
-keypass <Keypassword> -keystore <keystore> -storepass <storepassword>
```

For example:

```
keytool -import -v -trustcacerts -alias appserbatch -file c:/Rockwell/BatchServer/
appserbatch.cer -keypass password -keystore c:/Rockwell/BatchServer/truststore.p12
 -storepass password
```

9. Define a Bean file for the configuration of the *sslContextParameters*.

```
@Bean
    public SSLContextParameters sslContextParameters() {
        final SSLContextParameters scp = new SSLContextParameters();
        final KeyStoreParameters ksp = new KeyStoreParameters();
        ksp.setResource(new File(this.trustStorePath).toURI().toString());
        ksp.setPassword(this.trustStorePassword);
        ksp.setType("PKCS12");
        final KeyManagersParameters kmp = new KeyManagersParameters();
        kmp.setKeyPassword(this.keyManagerPassword);
        kmp.setKeyStore(ksp);
        final TrustManagersParameters tmp = new TrustManagersParameters();
        tmp.setKeyStore(ksp);
        scp.setKeyManagers(kmp);
        scp.setTrustManagers(tmp);
        return scp;
    }
```

10. Use the settings above in the HTTP component.

```
<to id="httpPostRequest" pattern="InOut"
uri="https:{{batchserver.url}}?sslContextParameters=#sslContextParameters&amp;
authMethod={{batchserver.authenticationMethod}}&amp;authUsername={{batchserver.user}}&a
mp;
authPassword={{batchserver.password}}"/>
```

TIP

Encrypt Passwords using the RunJasyptPasswordUtil script present in *<EIHub_Install>\installers\EIHub\Utility-Tools\encryption-tool-<FTEI version>.0xx.zip* file. Encrypted passwords in the property file are decrypted using the properties present in the *application.properties*:

```
jasypt.encryptor.password==${ACTIVEMQ_ENCRYPTION_PASSWORD}
jasypt.encryptor.algorithm=PBEWITHHMACSHA256ANDAES_256
```

3. Define the bean required for ActiveMQ integration, as shown in the following script:

```
    @Bean
    public ActiveMQSslConnectionFactory activeMQSslConnectionFactory() throws Exception {
        ActiveMQSslConnectionFactory factory = new ActiveMQSslConnectionFactory(url);
        if (trustStore != null && !trustStore.isEmpty()) {
            factory.setTrustStoreType("pkcs12");
            factory.setTrustStore(new File(trustStore).toURI().toString());
            factory.setTrustStorePassword(trustStorePassword);
        }
        factory.setUserName(user);
        factory.setPassword(password);
        factory.setTrustedPackages(Arrays.asList(trustedPackages.split(",")));
        return factory;
    }
}
```

Figure 35: Bean for ActiveMQ integration (create DCS batch touchpoint)

Notice the use of the application property *activemq.brokerURL* as an URL.

You also may need to set the pre-fetch limit for the consumers in order to have efficiently performing concurrent consumers. See the Apache ActiveMQ Getting Started Document [C2] (page 3).

4. Define the SSL context parameters Bean, as shown in the following script:

```
@Bean
public SSLContextParameters sslContextParameters() {
    final SSLContextParameters scp = new SSLContextParameters();
    final KeyStoreParameters ksp = new KeyStoreParameters();
    ksp.setResource(new File(this.trustStorePath).toURI().toString());
    ksp.setPassword(this.trustStorePassword);
    ksp.setType("PKCS12");
    final KeyManagersParameters kmp = new KeyManagersParameters();
    kmp.setKeyPassword(this.keyManagerPassword);
    kmp.setKeyStore(ksp);
    final TrustManagersParameters tmp = new TrustManagersParameters();
    tmp.setKeyStore(ksp);
    scp.setKeyManagers(kmp);
    scp.setTrustManagers(tmp);
    return scp;
}
```

Figure 36: Defining the SSL context parameters (create DCS batch touchpoint)

5. Define the route Templates and configure the route starting with the ActiveMQ endpoint from which you need to read the *CreateDCSBatch.xml* message as shown in the following script:

```
<routeTemplates xmlns="http://camel.apache.org/schema/spring">
<routeTemplate id="createDCSBatchTemplate">
<route id="createDCSBatch">
<description>Read createDCSBatch BML message from queue,
transform into an HTTP request to RestFul Batch Service
and process the response.</description>
<from id="getCreateDCSBatchMessageFromQueue"
uri="activemq:queue:DCSRequest_{{DCS_Adapter}}_CreateDCSBatch?exchangePattern=InOut"/>
```

Figure 37: Defining an ActiveMQ endpoint (create DCS batch touchpoint)

Note the use of the *DCS_Adapter* application property.

6. Add the transformation of the incoming payload to a JSON payload. The processing is done via the *CreateDCSBatch.groovy* script.

```
 <to id="transformToPostMessage" pattern="InOut"
uri="language:groovy:resourceUri?script={{groovyScript.path}}CreateDCSBatch.groovy"/>
```

Figure 38: Processing via the Groovy script (create DCS batch touchpoint)

The Groovy script contains the logic to transform the XML payload into a JSON object and use a RESTFul service call using the HTTP component:

```
import groovy.xml.*;
import groovy.json.*;
import org.apache.camel.*;

def getParameterValue(xmlParameter) {
    if (xmlParameter.ValueNumeric.text()) {
        return xmlParameter.ValueNumeric.text()
    }
    if (xmlParameter.ValueString.text()) {
        return xmlParameter.ValueString.text()
    }
    if (xmlParameter.ValueInteger.text()) {
        return xmlParameter.ValueInteger.text()
    }
    if (xmlParameter.ValueBoolean.text()) {
        return xmlParameter.ValueBoolean.text()
    }
    if (xmlParameter.ValueDateTime.text()) {
        return xmlParameter.ValueDateTime.text()
    }
    return "";
}

def processCreateDCSBatch = new
XmlSlurper(false,false).parseText(request.getBody().toString());

def sender = processCreateDCSBatch.ApplicationArea.Sender.text();
def receiver = processCreateDCSBatch.ApplicationArea.Receiver.text();

def batchID = processCreateDCSBatch.DataArea.BatchID.text();
def recipeID = processCreateDCSBatch.DataArea.MasterRecipeID.text();
def scaledSize = processCreateDCSBatch.DataArea.ScaledSize.text();
```

```
def formulaID = processCreateDCSBatch.DataArea.FormulaID.text();
def description = processCreateDCSBatch.DataArea.Description.text();

exchange.setProperty("sender", sender) exchange.setProperty("receiver", receiver)

//Steps / Equipment requirements def stepsSequence = []

processCreateDCSBatch.DataArea.ControlRecipe.EquipmentRequirements.each { elt ->
elt.children().each { equipmentRequirement ->
        def eqmRequirement = [
            name : equipmentRequirement.Constraint.text(),
            value : equipmentRequirement.ID.text()]

        stepsSequence.add(eqmRequirement)
    }
}

//process parameters
def parameterSequence = []
processCreateDCSBatch.DataArea.ControlRecipe.Parameters.each { elt ->
elt.children().each { parameter ->
        def param = [
            name : parameter.ID.text(),
            value : getParameterValue(parameter)]
        parameterSequence.add(param)
    }
}

def materialSequence = []

def jsonBuilder = new JsonBuilder()
jsonBuilder.createValues {
        recipe recipeID
        batchid batchID
scale scaledSize
        description: description
        steps stepsSequence
        parameters parameterSequence
        materials materialSequence
}

request.removeHeaders("JMS*")
request.removeHeaders("breadc*")
request.removeHeaders("User-Agent*")
request.setHeader(exchange.HTTP_METHOD, "POST")
request.setHeader(exchange.CONTENT_TYPE, "application/json")

return jsonBuilder.toString()
// return message provides the body of the http request
```

TIP
It is important to understand how the HTTP POST request is generated.

7. Issue the RESTFul service HTTP POST request and unmarshall the response.

```
  <to id="httpPostRequest" pattern="InOut"
uri="https:{{batchserver.url}}?sslContextParameters=#sslContextParameters&amp;
authMethod={{batchserver.authenticationMethod}}&amp;authUsername={{batchserver.user}}&amp;
authPassword={{batchserver.password}}"/>
<unmarshal id="UnmarshallJsonData">
    <json library="Gson" prettyPrint="false"/>
</unmarshal>
```

Figure 39: Issuing and unmarshalling the request (create DCS batch touchpoint)

The implementation of the RESTful service provides a response payload as a JSON object.

8. Transform the received JSON object into the appropriate XML message. This is done via the *AcknowledgeCreateDCSBatch.groovy* script.

```
    <to id="transformHTTPResponseMessage" pattern="InOut"
uri="language:groovy:resourceUri?script={{groovyScript.path}}AcknowledgeCreateDCSBatch.groovy"/>
```

Figure 40: Transforming the received JSON object (create DCS batch touchpoint)

```
import groovy.xml.*;
import groovy.json.*;

def httpRequestResponse = response.getBody();
def receiver = exchange.getProperty("sender")
def sender = exchange.getProperty("receiver")

def writer = new StringWriter();
def xml = new MarkupBuilder(writer);
xml.AcknowledgeCreateDCSBatch('xsi:schemaLocation':'http://www.rockwell.com/mes/dcs/ifc
../../../main/xsd/mes-dcs-interface.xsd',
'xmlns':'http://www.rockwell.com/mes/dcs/ifc',
        'xmlns:xsi':'http://www.w3.org/2001/XMLSchema-instance',
'schemeVersionID':'1.0') {
        ApplicationArea {
            Sender(receiver)
            Receiver(sender)
        }
        DataArea {
            if (httpRequestResponse['status'] == '0')
            { ResponseCriteria(actionCode:'Accepted')
        } else {
                    ResponseCriteria(actionCode:'Rejected',
                        httpRequestResponse['id'])
                }
                InternalBatchID(httpRequestResponse['status'] == '0'?
httpRequestResponse['createid']:'')
        }
        OtherInformations {
                OtherInformation {
                        ID ('UniqueBatchID')
                        ValueString ( httpRequestResponse['status'] == '0'?
httpRequestResponse['creatid']:'')
                }
        }
```

```
}
return writer.toString()
```

9. Add exception management in order to generate a reject message. The generation of the reject message is done via the *RejectCreateDCSBatch.groovy* script.

```
<onException id="createDCSBatch_multipleCauseException">
    <exception>java.lang.Exception</exception>
    <handled>
        <constant>true</constant>
    </handled>
    <to id="generateCreateDCSBatchRejectResponse"
        pattern="InOut" uri="language:groovy:resourceUri?script={{groovyScript.path}}RejectCreateDCSBatch.groovy"/>
</onException>
```

Figure 41: Adding exception management (create DCS batch touchpoint)

If an exception is thrown during the life of the route's exchange, a reject message will be generated via the Groovy script.

```
import groovy.xml.*;
import groovy.json.*;

def receiver = exchange.getProperty("sender") != null ? exchange.getProperty("sender"):
"UNKNOWN"

def sender = exchange.getProperty("receiver") != null ? exchange.getProperty("receiver"):
"UNKNOWN"

def cause = exchange.getProperty(org.apache.camel.Exchange.EXCEPTION_CAUGHT, Exception.
class);

def excClassName = cause.getClass().getName()
def errorMessage = excClassName + ': Please check log files on MessageBroker for more
details'

def writer = new StringWriter();
def xml = new MarkupBuilder(writer);
xml.AcknowledgeCreateDCSBatch('xsi:schemaLocation':'http://www.rockwell.com/mes/dcs/ifc
../../../main/xsd/mes-dcs-interface.xsd','xmlns':'http://www.rockwell.com/mes/dcs/ifc',
            'xmlns:xsi':'http://www.w3.org/2001/XMLSchema-instance', 'schemeVersionID'
:'1.0') {
            ApplicationArea {
                Sender(receiver)
                Receiver(sender)
                }
                DataArea {
                    ResponseCriteria(actionCode:'Rejected',
                    errorMessage)
                    InternalBatchID('')
                }
}
return writer.toString()
```

Chapter 5

Externalization of Configuration

Overview

Configurations in the installed application can be added or modified for scripts, camel routes, email templates, and application property files by adding or modifying the externalized files in the application installation directory *(EIHub-app-<valuepack>-<FTEI version>.0xx).* The directory structure can be explained as follows:

- *bin* - contains the scripts to start the EIHub as a standalone application.
- *certs* - contains SSL certificates and keystores used by the application.
- *config* - contains application property and route template builder files.
- *eihub-routes-config* - contains route template xml files.
- *email-templates* - contains email template files for failure notification emails.
- *groovy-scripts* - contains all the groovy script files used in route processing.
- *lib* - contains the jar archives used by the standalone EIHub application.
- *log* - contains log files of the EIHub application.

TIP

email-templates are not provided out of the box for dcs value pack.

Externalized Configuration for Routes

With externalized Camel configuration, comes an advantage of making changes in the XML configuration for routes and route builder without doing any Java code changes and compiling the code again.

Types of Camel Configuration Files

There are two types of XML files present for Camel configuration:

1. **Route template definitions**

   You can define multiple routes using <*route*> tag inside <*routeTemplate* > for specific task. Similarly a < *routeTemplates*> can have multiple < *routeTemplate*>.

```
<routeTemplates xmlns="http://camel.apache.org/schema/spring">
    <routeTemplate id="getDCSBatchValuesTemplate">
        <route id="getDCSBatchValues">
            <description>Read getDCSBatchValues BML message from queue, transform into a
SQL request, execute the SQL retrieval and transform the resultset response into the BML
response</description>
            <from id="getDCSBatchValuesMessageFromQueue"
uri="activemq:queue:DCSRequest_{{DCS_Adapter}}_GetDCSBatchValues?exchangePattern=InOut"
/>
            <log loggingLevel="DEBUG" message="${body}"/>
            <to id="transformToBatchValuesQuery" pattern="InOut"
uri="language:groovy:resourceUri?script={{groovyScript.path.dcs}}GetDCSValuesBuildQuery
.groovy"/>
            <log loggingLevel="DEBUG" message="${body}"/>
            <to id="retrieveBatchValuesBySQL" pattern="InOut" uri="sql:select
1?exchangePattern=InOut"/>
            <log loggingLevel="DEBUG" message="${body}"/>
            <to id="transformBatchValuesResultSet" pattern="InOut"
uri="language:groovy:resourceUri?script={{groovyScript.path.dcs}}AcknowledgeGetDCSValue
s.groovy"/>
            <onException id="getDCSBatchValues_multipleCauseException">
                <exception>java.lang.Exception</exception>
                <handled>
                    <constant>true</constant>
                </handled>
                <to id="generateDCSBatchValuesRejectResponse"
                    pattern="InOut"
uri="language:groovy:resourceUri?script={{groovyScript.path.dcs}}RejectGetDCSValues.gro
ovy"/>
       <log message="${body}"/>
            </onException>
            <log message="${body}"/>
         </route>
    </routeTemplate>
</routeTemplates>
</routeTemplates>
```

2. **Routes template includer also known as template builder**

   Below is the structure of the XML used for defining routes inclusion or overall route builder file for application. The *<templatedRoutes>* tag contains all the *<templatedRoute>* necessary to include in an application. Application considers only those routes defined in this file for management.

```
<templatedRoute routeId="jmsErrorMsgCaught"
routeTemplateRef="jmsErrorMsgCaughtTemplate">templatedRoute-jmsErrorMsgCaught</template
dRoute>
```

Configuration Details

Below is the property that helps with including external files at runtime into the application.

```
camel.springboot.routes-include-pattern=file:src/dist/dcs/eihub-routes-config/**/*.xml,
file:src/dist/dcs/config/dcs-route-builder.xml

file:src/dist/dcs/eihub-routes-config/**/*.xml : This refers to the path where the
application should look for route.
file:src/dist/dcs/config/dcs-route-builder.xml : This is the route builder file where we
define all the routes to be included in the application.
```

Adding New Routes

To add a new route, proceed as follows:

1. Create an XML file that contain the definition of route with proper *routeTemplateRef* ID.
2. Add above route in the route builder file with the *routeId* and provide *routeTemplateRef* as mentioned in step 1.

```
<templatedRoute routeId="getDCSAlarmEvents"
routeTemplateRef="getDCSAlarmEventsTemplate">templatedRoute-getDCSAlarmEventsTemplate</
templatedRoute>
```

Externalized Configuration for Application Properties

The following are the application properties files that are used for configuration.

- ***application.properties*** - contains common properties for all valuepacks.

application.properties

```
#profile
spring.profiles.active=dcs

#log-level
logging.level.root=info
trace=false

#server-port
server.port=8443

#hazelcast
spring.hazelcast.config=file:src/main/resources/config/hazelcast.xml

#static resource path
spring.web.resources.static-locations=classpath:/hawtio-static/

#logging level for hazelcast
logging.level.com.hazelcast=ERROR

#I18n property
spring.messages.basename=lang/messages

#disable whitelabel page
server.error.whitelabel.enabled=false

#jasypt properties
jasypt.encryptor.password=${ACTIVEMQ_ENCRYPTION_PASSWORD}
jasypt.encryptor.algorithm=PBEWITHHMACSHA256ANDAES_256

#route-path
camel.springboot.routes-include-pattern=

#hawtio configuration
camel.springboot.name=
management.endpoints.web.exposure.include=hawtio,jolokia
management.endpoints.web.base-path=/
camel.springboot.backlog-tracing=true
spring.jmx.enabled=true

#hawtio ssl configuration
server.ssl.key-store=file:src/main/resources/certs/<keystore>.p12
server.ssl.key-store-password=
server.ssl.key-store-type=
server.ssl.key-alias=
server.ssl.key-password=

#ldap configurations
ldapUrl=ldap://<hostname>:<port>/dc=<value>,dc=<value>
userDnPatterns=uid={0},ou=public

#JMS Provider Connection URL
spring.activemq.broker-url=failover\://(ssl\://<hostname>\:<port>)?startupMaxRecon
spring.activemq.user=
spring.activemq.password=
spring.activemq.trust-store=file:src/main/resources/certs/<truststore>.p12
spring.activemq.trust-store-password=

#properties to trust the packages
spring.activemq.packages.trusted=java,com.rockwell,com.datasweep,sun,com.osiso
```

Figure 42: Properties defined in the application.properties file

- ***application-< valuepack >.properties*** - contains valuepack specific properties.

```
#import custom property file
spring.config.import=custom-application-${spring.profiles.active}.properties

#route-path
camel.springboot.routes-include-pattern=file:src/dist/dcs/eihub-routes-config/**/*.xml,file:src/dist/dcs/config/dcs-route-builder.xml

#Camelcontext name
camel.springboot.name=pharmasuite-plantpax-msb-context

#properties
GMPAlarm.severityLevel=0
DCS_Adapter=PlantPAx

#Groovy file path
groovyScript.path.dcs=file:src/dist/dcs/groovy-scripts/

#API configuration
getdcsvalues.numericScale=3
batchserver.url=https://<hostname>:<port>/batches
batchserver.user=
batchserver.password=
batchserver.authenticationMethod=Basic

#API truststore configuration
trustStore.file=file:src/dist/dcs/certs/<filename>.p12
keyManagers.keyPassword=
trustStore.password=

#database configuration
spring.datasource.url=jdbc:sqlserver://<hostname>:<port>;databaseName=<DBName>;trustServerCertificate=true;
spring.datasource.username=
spring.datasource.password=
spring.datasource.driverClassName=com.microsoft.sqlserver.jdbc.SQLServerDriver
```

Figure 43: Properties defined in the application-< valuepack >.properties file

- ***custom-application-< valuepack >.properties*** - any existing property that needs be customized should be configured here using the same property key as in the application property file.

Index